Chapter 3

Operating Systems

Operating system (OS):

- A collection of programs that **manages resources** of a computer, such as
  - processors
  - memory
  - input/output devices
- ... like the conductor of an orchestra.
- A **virtual machine** that lets a user accomplish tasks that would be difficult to perform directly with the underlying actual machine.
  - graphical user interface
  - virtual memory: provide more memory than in RAM
  - multiprogramming: seemingly run more than one program at a time

3.1 Evolution of Operating Systems

First Generation OS (1945-1955)

The only “operating system” was a person. All machine operation was “hands on”.

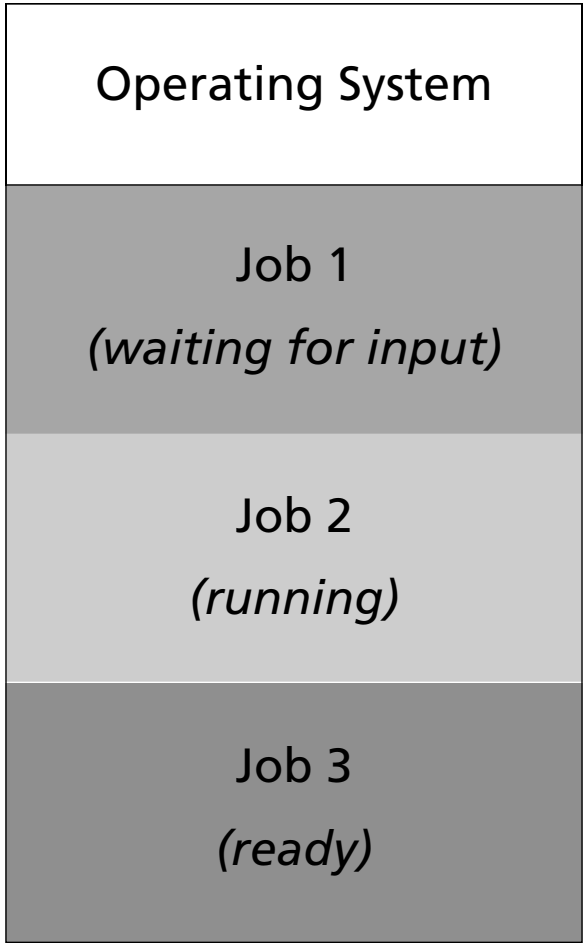
Second Generation OS (1955-1965)

Batch operating systems:

- A computer operator takes care of the system administration.
- The operator collects a “batch” of programs from several programmers, feeds the programs into the computer, and hands out the printed results back to the programmers.
- Today, batch operations are performed automatically.
- Compute jobs line up in a **FIFO job queue** (first in, first out).

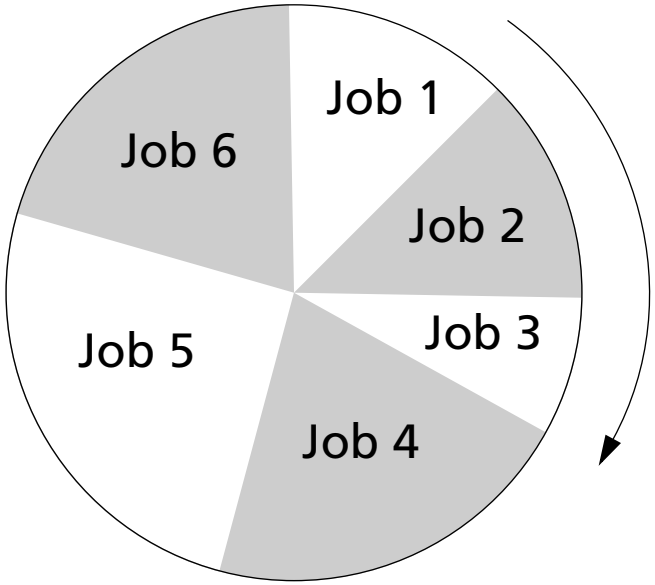
Multiprogramming OS

Switching between many different programs in memory (RAM)



Multiprogramming enables

- **time sharing**:
  - divides processor time up into slices
  - the slices are divided "fairly" among competing jobs



- **interactive processing**:
  - the user does not have to wait for one process to be finished until the next process (program, action) can be started

Multiuser Operating Systems

Many different users can share the same machine through time sharing and multiprogramming (e.g., UNIX, MacOS X, Windows NT).

The OS divides its system time into time slices (milliseconds).

This gives each user the illusion to have his/her own machine.

The efficiency of a time-sharing system depends on

- the speed of the processor
- the length of the time slices
- how many users perform operations that require a full time slice.

Multiuser OSs are mainly used in distributed, rather than centralized, environments, where several machines share resources over a network.

Single-User OS and Multiprogramming

- Cooperative multitasking:

  Switching between programs only if those programs explicitly cooperate.

- Preemptive multitasking:

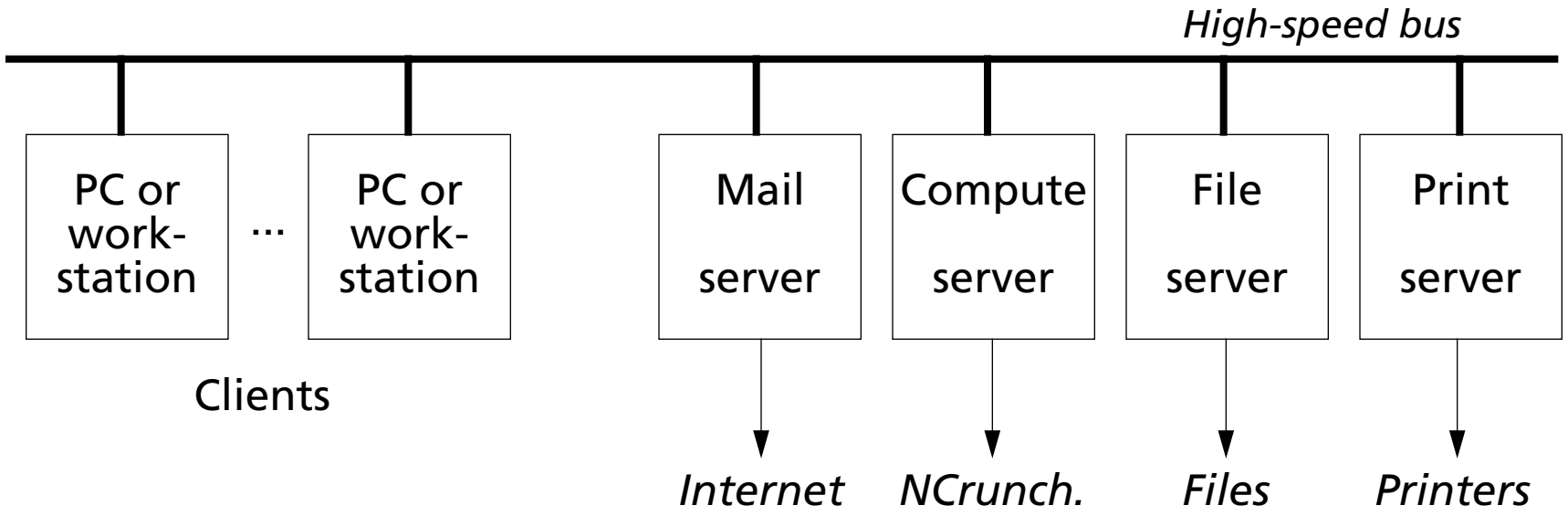
  Lets the OS switch between the tasks at its own discretion.

  $\Rightarrow$ Perform different tasks at the “same time” (printing, compiling, searching on the web, ...)

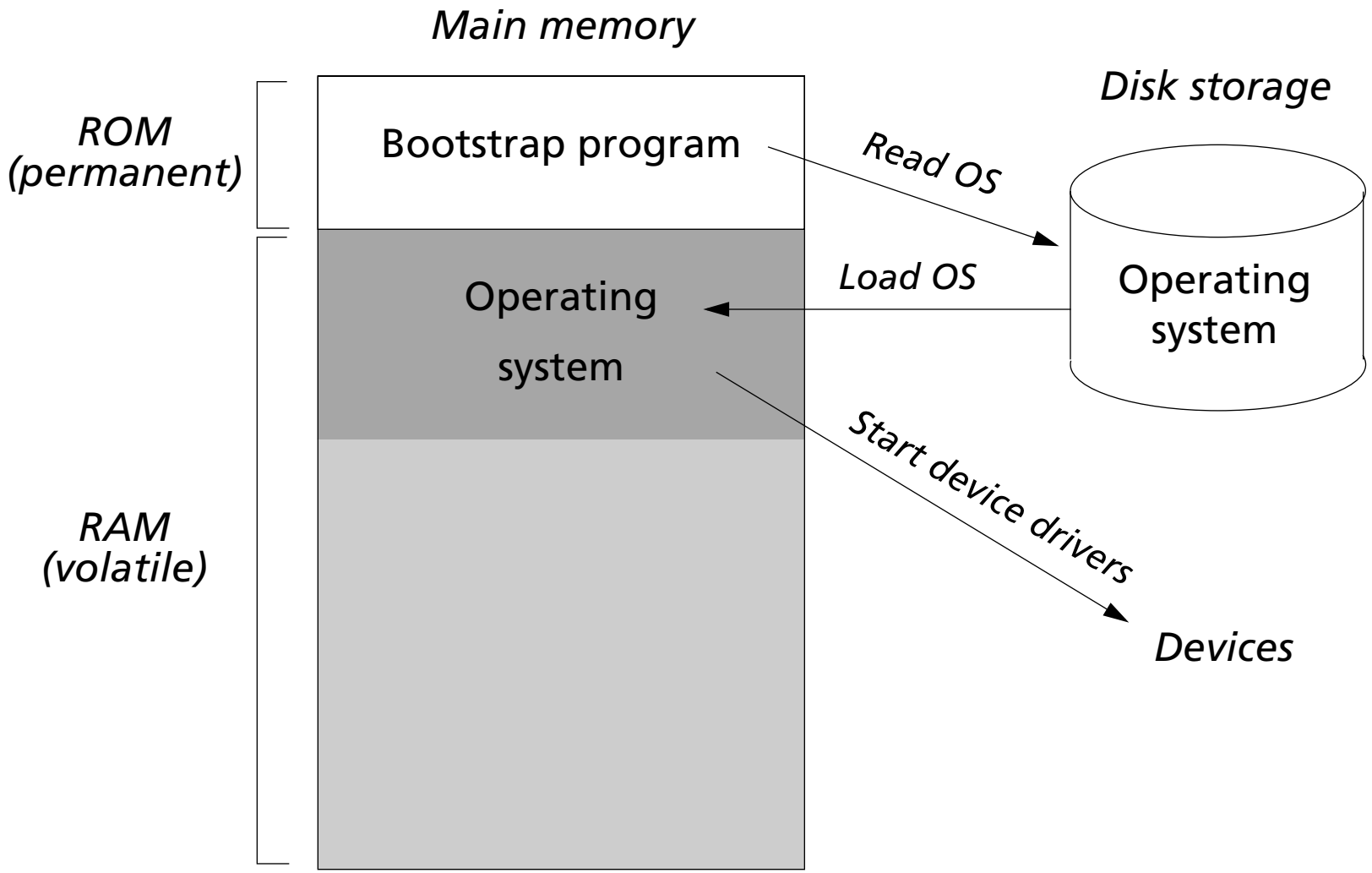
Network Operating Systems

Sharing of more expensive peripheral devices (in a LAN or WAN):

- laser printers
- 3D plotters
- tape backup units
- fast number-crunching machines
- special-purpose software packages



3.2 Booting an Operating System



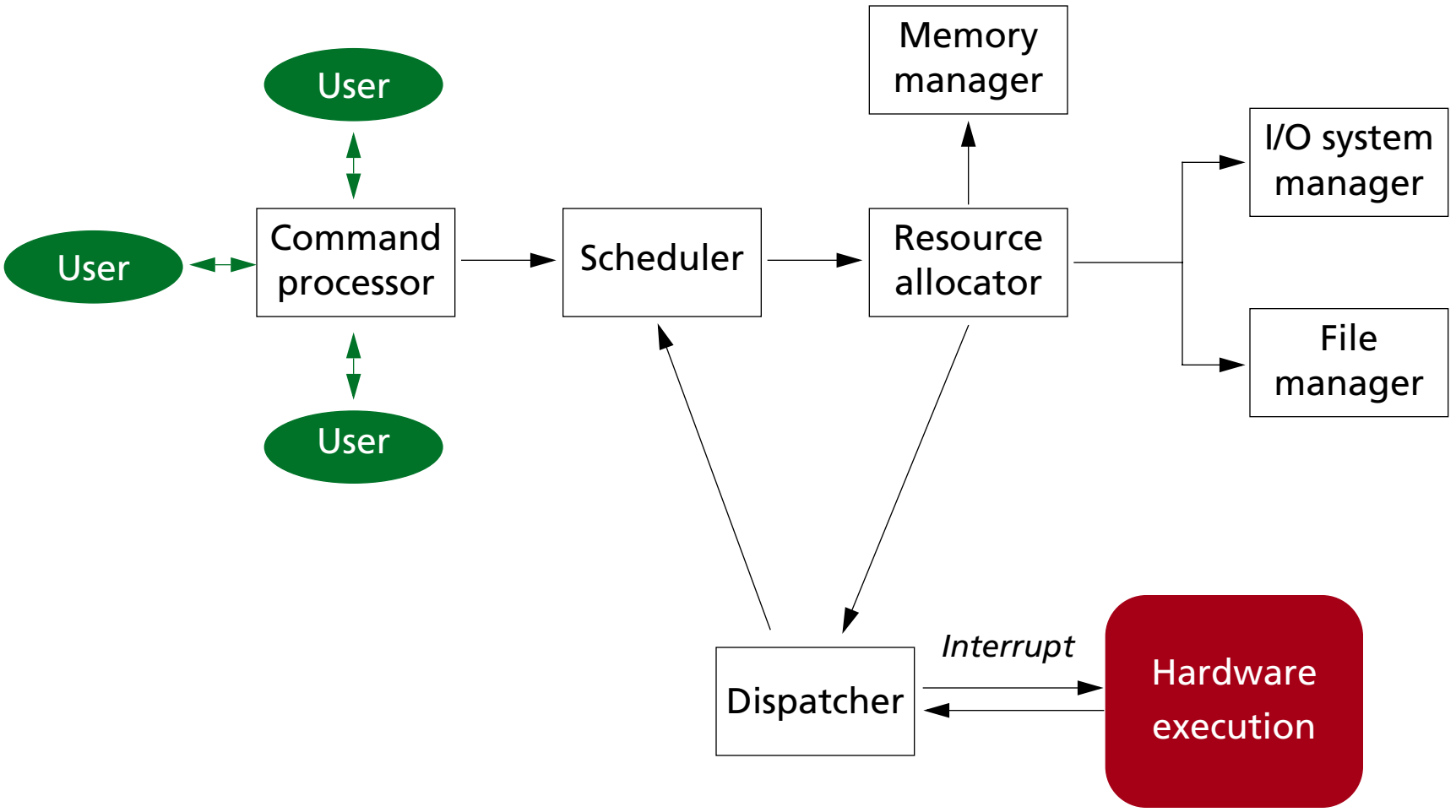
Starting up an operating system

- Read boot strap program from ROM:

  $\Rightarrow$ enables access to floppy and hard drives

  $\Rightarrow$ look for the core OS (DOS: **`COMMAND.COM`**)
- Load the core OS into RAM.
- Run a sequence of jobs in batch mode.
  - DOS: **`CONFIG.SYS`** and **`AUTOEXEC.BAT`**
  - Windows: **`WIN.INI`**
  - MacOS: selected extensions
  - UNIX: **`.login`**, **`.profile`**, **`.cshrc`**
- Start a graphical user interface (GUI)
- Loop forever waiting for input / interaction with the GUI.

3.3 Operating System Architecture



Command processor:

The interface that interacts with one or more users, watching the keyboard, mouse, and any other input devices attached to the machine.

Receives commands whenever an input device notifies it about an **event**, for example, when a user enters a new line in a shell or clicks on a mouse button.

Scheduler:

If the event turns out to be a **request to run a program**, the command processor asks the scheduler to arrange for the execution of programs.

- Place the program in a job queue.
- Create a process to execute the program.

A **process** is an active copy of a program

- that has been loaded into memory (RAM)
- with its own program counter indicating the next instruction to execute.

Each process creates its own **context** (= process state).

There may be more than one process for the same program (e.g., start two different editor programs).

Resource Allocator

The scheduler invokes a resource allocator, which makes sure that each process has **secondary resources** that it may need — memory, files, peripheral devices.

Memory Manager

The memory manager coordinates the use of the machine's memory.

- It keeps track of which areas of memory are being used by which processes.
- It may anticipate how much memory each process will need.
- It may provide virtual memory, where "swap files" on a harddrive are used to extend the RAM.

I/O System Manager

The I/O system manager coordinates the assignment of peripheral devices to processes, through specific device drivers.

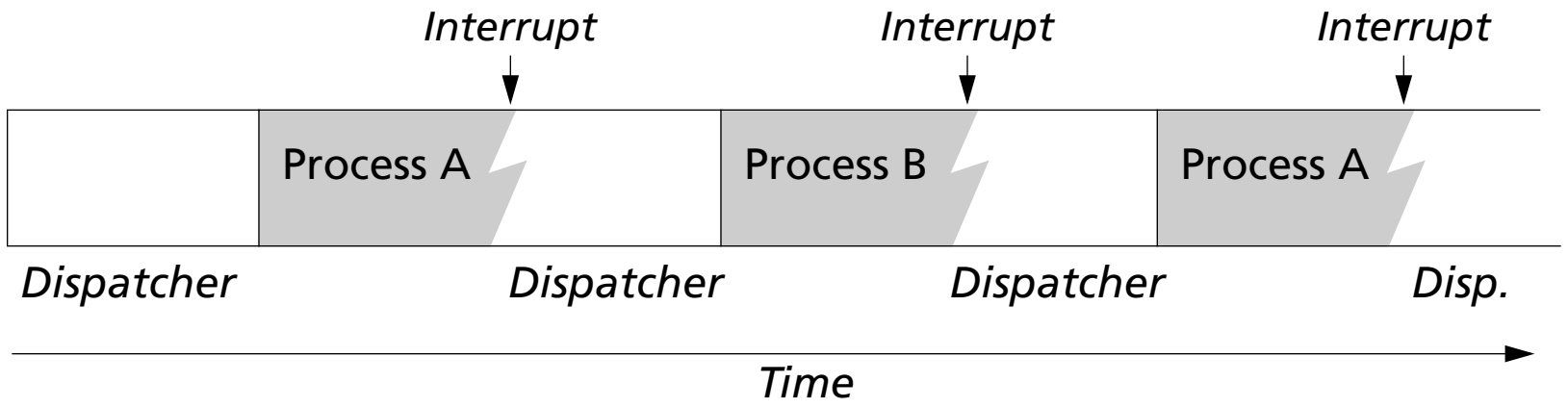
File Manager

The file manager keeps track of information about files on the disks:

- protects against unauthorized file access
- supports sharing of file resources
- helps organizing files into folders

Dispatcher

The dispatcher monitors processes and decides when to switch execution from one process to another.



When a process completes ...

⇒ the dispatcher reports back to the scheduler,
⇒ the scheduler notifies the resource allocator,
⇒ the resource allocator releases any resources held by that process,
⇒ the scheduler then reports completion of the process to the command processor,
⇒ the command processor informs the user.

3.4 References

- G. Blank and R. Barnes, *The Universal Machine*, Boston, MA: WCB/ McGraw-Hill, 1998. Chapters 10.1 through 10.3.